Productivity Engineering in the UNIX† Environment

UNIX Consultant: A Progress Report

Technical Report

S. L. Graham

(415) 642-2059

"The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government."

Contract No. N00039-82-C-0235

November 15, 1981 - December 30, 1985

Arpa Order No. 4031

†UNIX is a trademark of AT&T Bell Laboratories

(1) (2)

Productivity Engineering in the UNIX† Environment

UNIX Consultant: A Progress Report

Technical Report

S. L. Graham

(415) 642-2059

Contract No. N00039-82-C-0235

November 15, 1981 - December 30, 1985

Arpa Order No. 4031

†UNIX is a trademark of AT&T Bell Laboratories

UNIX Consultant: A Progress Report

Robert Wilensky
Division of Computer Science
University of California, Berkeley

We have been engaged in the construction of a system called UC, for *UNIX Consultant*. UC is designed to be an automated consultant that converses in natural language with naive users to help answer their questions about the UNIX operating system. The intent is to provide a system that allows a naive user to ask questions about terminology, about command names and formats, for information concerning plans for doing things in UNIX, and for assistance in debugging problems with UNIX commands. UC should respond by explaining terminology, providing command names and describing their format, filling in the details of a user plan, suggesting plans to achieve goals, or engaging a user in a dialogue by requesting more information.

To achieve this goal requires research into basic issues in natural language processing and common sense reasoning. Our research views the user as a planning agent who has goals implicity or explicitly expressed by his or her utterance. It is UC's task to determine the user's goal and to aid the user by providing information for the achievement of those goals.

In our previous work, we had implemented a prototype version of this system. Much of our research work has addressed shortcomings with the technology that limited the success of our initial efforts.

Probably the most fundamental problem is one of knowledge representation. Weaknesses in the knowledge representation scheme we were employing, and, indeed, in knowledge representation schemes in general, prevented our system from having the flexibility and extensibility we sought. To rectify this problem, we developed a new knowledge representation scheme called KODIAK. KODIAK is described in Wilensky [11]. Some important characteristics of KODIAK are: It clarifies the semantics of slots; it is uniform, and applies to any number of semantic domains; it based on relations, and in particular, has a small set of primitive epistemological relations and allows for the creation and definition of new relations, and it has a canonical form.

Perhaps the most interesting aspect of KODIAK is the introduction of "non-truth conditional" representation entities. We call such entities *views*. The idea behind views is that one concept can be thought of in terms of another. This viewing of another concept creates a new concept. It appears as if the introduction of such entities solves a number of long standing representational issues.

For example, one problem is that most systems provide one category called **Person**, and other called **Physical-Object**, and presume that these are in fact distinct. This is a problem because in many case, we want people to inherit

properties of physical objects, even though they are not properly classified as such. We address this issue in KODIAK by asserting that **Person** is a kind of **Living-Thing**, but, in addition, we also assert that it is possible to **VIEW** a **Person** as a **Physical-Object**. Morever, the **VIEW**of **Person** as a **Physical-Object** is itself another concept. Namely, it is the concept **Body**. Thus we can allow **Person** to share some of the properties of **Physical-Object** without it properly being represented as one. The importance of views is that the allow the flexibility of viewing a (possible defined) concept as something other than its "ordinary" interpretation. We believe that views constitutes a major finding in the area of knowledge representation.

Views appear to be particularly important in representing knowledge about language. Jacobs [7] points out that many otherwise unstatable linguistic regularities can be captured using views. For example, there are many cases like "John took a punch from Bill", "Bill gave John a punch", and "John gave Mary a kiss" in which it appears that a person being acted upon can be talked about in terms of giving and taking. Our solution is to represent a "being acted upon as transfer" view, for the purposes of linguistic expression. That is, an object being acted upon can be viewed as transferring the action to that object. Once this view is represented, knowledge about the linguistic expression of transferring (e. g., that certain transfers can be expressed using the words "give" and "take") can be employed to express certain kinds of non-transferring actions that act upon some object.

We are completing a second version of UC that incorporates some of these aspects of knowledge representation. KODIAK was used as the representation language for the construction of UC's knowledge-base. The knowledge-base included knowledge about linguistic patterns, UNIX concepts and commands, user and UC goals, and planning. During a course of an interaction with a user, UC adds new knowledge to the knowledge-base in the form of instantiations of the general knowledge already contained in its knowledge-base. The addition of new knowledge serves as the method of communication between the various components. Its semantics is defined by its relationship to the rest of the knowledge-base. Because it creates the form of communication between subprocesses of UC, the construction of the knowledge-base more clearly focuses the responsibilities of the various components of UC.

This version of UC can handle over 50 different question types. Among other things, it is able to generate examples where appropriate. Here is some sample actual I/O:

```
=> (UC)
Welcome to UC (Unix Consultant) version 3
        Please type in your questions about Unix in English.
        If you want information about how to use this program type 'help'.
        To leave, just type 'exit'.
```

User: Can you tell me how to move a file?
To move a file to another file, use mv.
For example, to move the file named paul to the file named peter, type mv paul peter.

User: Do you know how to delete a file named dekai?
To delete the file named dekai, type rm dekai.

User: Do you know how to delete a file?
To delete a file, use rm.
For example, to delete the file named lisa, type rm lisa.

User: How can I delete the file charlie?
To delete the file named charlie, type rm charlie.

User: How can I delete the file named tony?
To delete the file named tony, type rm tony.

User: How can I get rid of the file named boob?
To delete the file named boob, type rm boob.

User: I want to delete a file named peter.
To delete the file named peter, type rm peter.

User: I want to delete a file.
To delete a file, use rm.
For example, to delete the file named jim, type rm jim.

User: I want to delete the file dave.
To delete the file named dave, type rm dave.

User: How can I add read permission to a file?
To add read permission to a file, use chmod.
For example, to add individual read permission to the file named charlie, type chmod u+r charlie.

User: How can I add the owner execute permission to a file?
To add individual execute permission to a file, use chmod.
For example, to add individual execute permission to the file named marc, type chmod u+x marc.

User: How can I change the group execute permission of a file?
To change the group execute permission of a file, use chmod.
For example, to remove group execute permission from the file named jim, type chmod g-x jim.

User: How can I change the group read permission of a file?

To change the group read permission of a file, use chmod.
For example, to remove group read permission from the file named george, type chmod g-r george.

User: How can I remove the group write permission from the file named bimbo?
To remove group write permission from the file named bimbo, type chmod g-w bimbo.

User: How do I change the protection of a file?
To change the permission of a file, use chmod.
For example, to remove individual read permission from the file named paul, type chmod u-r paul.

User: How can I compact a file?
To compact a file, use compact.
For example, to compact the file named peter, type compact peter.

User: How can I create a directory?
To create a directory, use mkdir.
For example, to create the directory named paul, type mkdir paul.

User: How can I find a file?
To find a file, use whereis.
For example, to find the file named charlie, type whereis charlie.

User: How can I find out how much disk space I am using?
To find out how much disk space you are using, type du.

User: How can I find out who is using the system?
To find out all the users on the system, type finger.

User: How can I find out wilensky's phone number?
To find out Wilensky's phone number, type finger wilensky.

We have also been developing additional kinds of planning mechanisms to support the reasoning component of our system. In particular, Richard Alterman [1-4] has been developing a technique called *adaptive planning*. The aim of adaptive planning is to borrow plans from related tasks and refit them to meet the demands of the current planning situation. For example, suppose UC has a plan for printing a file on the imagen using the UNIX 'lpr' command, by appending the argument '-Pip". An adaptive planner can re-use this plan in order to construct a plan for deleting a file from the imagen queue using the 'lprm' command by appending the argument '-Pip".

Alterman suggests several distinguishing features of adaptive planning. The first of these is that adaptive planning makes the background knowledge associated with an old plan explicit. By making the content and organization of the

background knowledge explicit, it becomes possible to re-interprete the plan for a wider variety of situations. A second important feature of adaptive planning is that it plans by situation matching. Rather than treating the old plan as a partial solution which is modified using weak (problem solving) methods, the old plan is used as a starting point from which the old and new situation are matched. In the course of the matching a new plan is produced. A third important feature of adaptive planning is that it can re-use old plans in circumstances where the planner does not have access to a general plan. As shown in the example above, adaptive planning is capable of taking a specific plan for accomplishing a specific set of goals, and refitting it to meet the demands of some new situation.

A prototype model of an adaptive planner is now under construction.

References:

[1] Alterman, Richard. Plexus: Networks for Adaptive Planning. In **Proceedings of the Second Workshop on Theoretical Issues in Conceptual Information Processing**. New Haven, CT. May, 1985.

[2] Alterman, Richard. Event Concept Coherence. To appear in **Advances in Natural Language Processing**. David Waltz (ed.). Lawrence Erlbaum Associates.

[3] Alterman, Richard. Towards a Theory of Adaptive Planning. To be appear in **Advances in Cognitive Science 1**, Noel Sharkey (ed.). Ellis Horowitz Associates.

[4] Alterman, Richard. Adaptive Planning: Refitting Old Planning Experiences to New Situations. Submitted to the Seventh Annual Conference of the Cognitive Science Society.

[5] Alterman, Richard. Summarization in the Small. To be submitted to **Cognitive Science**.

[6] Jacobs, Paul. Efficiency and Flexibility in a Language Generator. To appear in **Computational Linguistics**.

[7] Jacobs, Paul. PHRED: A Generator for Natural Language Interfaces. Berkeley Computer Science Division Report No. UCB/CSD85/198. January 1985.

[8] Jacobs, Paul. A Knowledge-Based Approach to Language Production. Berkeley Computer Science Division Report No. UCB/CSD86/254. August

1985.

[9] Jacobs, Paul and Rau, Lisa. Ace: Associating Language with Meaning. In **ECAI-84: Advances in Artificial Intelligence**. T. O'Shea (ed.). Elsevier Science Publishers B. V. (North Holland) ECCAI, 1984.

[10] Rau, Lisa. The Understanding and Generation of Ellipses in a Natural Language System. Berkeley Electronic Research Laboratory Memorandum, Fall 1984.

[11] Wilensky, Robert. KODIAK: A Knowledge Representation Language. **Proceedings of the 6th National Conference of the Cognitive Science Society.** April 1984.